PASSIVE ACQUISITION OF CLIPS RULES*

Vincent J. Kovarik Jr.

Software Productivity Solutions, Inc.
122 North 4th Avenue
Indialantic, FL 32903
(407) 984-3370

Abstract The automated acquisition of knowledge by machine has not lived up to expectations and knowledge engineering remains a human intensive task. Part of the reason for the lack of success is the difference in cognitive focus of the expert. The expert must shift his or her focus from the subject domain to that of the representation environment. In doing so, this cognitive shift introduces opportunity for errors and omissions. This paper presents work which observes the expert interact with a simulation of the domain. The system logs changes in the simulation objects and the expert's actions in response to those changes. This is followed by the application of inductive reasoning to move the domain-specific rules observed to general domain rules.

INTRODUCTION

The acquisition and application of complex domain knowledge remains a human intensive task. Efforts to directly elicit such knowledge from experts has not lived up to expectations and current knowledge acquisition tools fall short of direct acquisition from the expert. To a large degree, this is because tools focus on providing the knowledge engineer with powerful abstractions for building the declarative structures for the domain but do not encompass an understanding of the acquisition process. Consequently, human knowledge engineers must still be relied upon to develop a comprehensive knowledge base. Even those domain experts that learn to use knowledge engineering tools, their success is limited because of the necessity to shift their cognitive focus between the domain to the knowledge engineering environment. This shift in context results in errors and omissions.

The goal of this effort is to develop an approach to the direct extraction of knowledge from experts by emulating the unobtrusive observation and elicitation activities of a human knowledge engineer, i.e. a Passive Knowledge Acquisition System (PaKAS). This effort is not, as one might first assume, a knowledge engineering environment. PaKAS is intended to be a proactive system which "observes" the expert in the performance of a task, elicits rationale and background knowledge from the expert regarding the reasoning and decision behind the actions

* This work was supported by NASA, Johnson Space Center, under the Small Business Innovative Research (SBIR) program, contract NAS9-1129.

taken, and directly generates a knowledge base from this information. This generated knowledge base could then be modified and edited using an existing knowledge engineering toolset.

The first phase of PaKAS was developed in CLOVER, an object-oriented knowledge representation system developed by the author. CLOVER is a hybrid system, borrowing concepts from (Levesque 1984). It provides the usual capabilities of a traditional frame system in the areas of default reasoning, inheritance, attribute daemons. It also supports method definition, a message passing protocol, method daemons, and other programming constructs.

Acquisition by Observation

The "observation" of an expert by an automated system must make some assumptions and concessions regarding the actual mode of observation. Observation of expert behavior is not a totally new concept (Wilkins et al 1987) but the approach used by PaKAS is different. Of course, technology is not at a state where a machine can "visually" observe an expert and identify the actions performed by the individual, the objects acted upon, and the state changes resulting from the actions. Consequently, PaKAS works with a simulated environment. The expert works with the environment interactively, without any knowledge of the observation activity. The domain simulation logs each action performed by the expert and state changes resulting from the action.

Rather than forcing the expert to think both as an expert and as a knowledge engineer, the goal of PaKAS is to allow the expert to perform within their domain without external distraction. During their performance, the passive acquisition system watches, observes, and "take notes" regarding the actions performed by the expert. Then, as a human knowledge engineer does, after the expert has completed the task, the system queries the expert for rationale, goals, and anticipated results of their actions during the simulation execution. This allows the expert to focus only on the task at hand, thereby reducing the potential of omissions and oversights.

PaKAS Domain

Various options were discussed for the domain for the Phase I effort. It was decided that the Vacuum Vent Line / Intelligent Computer Aided Training tool (VVL/ICAT) would be used as the domain simulation. VVL/ICAT was developed to help train individuals in the concepts and troubleshooting of a vacuum system. It is constructed in Turbo C[1] and CLIPS.[2] Turbo C provides the basic user interface capabilities, along with the MetaWindows and Icon Tools support packages, and CLIPS provides the simulation capability. VVL/ICAT runs on an IBM PC or clone with a color graphics adapter and mouse.

After some experimentation with the VVL/ICAT tool, it became apparent that, although it was an excellent tool for building a vacuum vent line model and helping the user develop an understanding of the domain, it was not a simulation model which could be used as a front end for PaKAS. In order to propagate changes along the VVL model, the user needed to explicitly select a menu item and choose the proper option. For PaKAS, automatic propagation of changes whenever the user made some change in the model's state was necessary.

[1]Turbo C is a trademark of Borland International, Inc.

[2]CLIPS is a forward chaining inference engine developed by NASA, Johnson Space Center.

To support the type of model interaction desired for PaKAS, an object-oriented model was developed within the knowledge representation system This model provides model-based reasoning via discrete-event qualitative simulation. Since PaKAS would have had the model objects and a representation of the domain for acquisition purposes anyway, the only additional capabilities developed were the model execution and user interface. VVL/ICAT was used as the method for model construction and validation. PaKAS then imports models developed using VVL/ICAT and assumes that they have been checked for correctness by VVL/ICAT prior to being loaded into PaKAS.

Figure 1 shows the architecture of the Phase I PaKAS system. The VVL/ICAT system is used to construct and debug a vacuum vent line model and save that model to disk. PaKAS then reads and interprets the VVL/ICAT file and instantiates the corresponding knowledge classes representing the model.

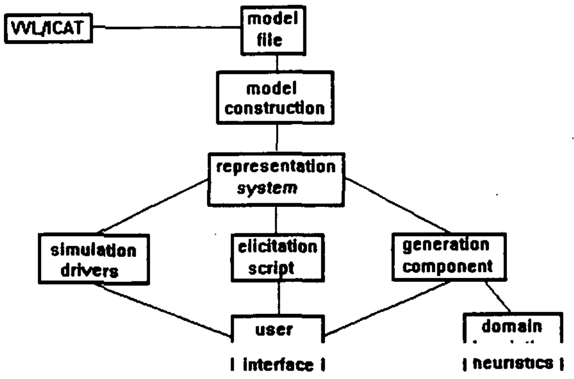

Figure 1. PaKAS Phase I Architecture

Domain Knowledge

Vacuum propagation begins by setting the state of the objects to a pre-vacuum condition. A vacuum is then propagated starting at the vacuum source(s) and continuing through the network of objects until a leaf node (i.e. a gauge or apparatus) or a closed valve is encountered. After the vacuum has been propagated, the leak is back-propagated to update objects affected by the leak condition. PaKAS uses a set of connection relations specified in the representation system to build a directed graph of the network topology from the vacuum sources outward. This represents the direction of vacuum propagation from a vacuum source.[3] Effectively, this collapses the connecting pipes into a single object between two or more model objects. The dependencies between the VVL simulation objects are shown in figure 2.

[3]Leaks are propagated in **both** directions along the network.

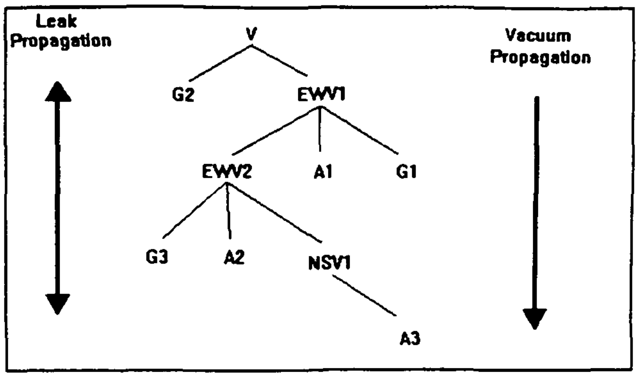

Figure 2. Model Dependencies

"V" is a vacuum source, "G*n*" represents some numbered gauge, "A*n*" represent some equipment or apparatus that uses the vacuum, and the "EWV*n*" and "NSV1" represent "east-west" and "north-south" valves, respectively.

The prototypical descriptions of the components are the description of the classes that make up the model. Each of the components is a descendent of the abstract class **model-component**. The model-component class defines basic methods and attributes common to all model objects. The lower-level classes (valves, gauges, apparatus, and vacuum) specialize this abstract class. The model-component object definition is shown below.

```
(defobject model-component physical-object
  :instantiable nil
  :attributes ((x-coord :cardinality 1 :default 0)
               (y-coord :cardinality 1 :default 0)
               (name :cardinality 1)
               (display-code :cardinality 1)
               (foreground-color :cardinality 1 :default 1)
               (background-color :cardinality 1 :default 0))
  :relations  ((connected-to :range model-component)))
```

The abstract class also defines basic simulation actions and value propagation routines for the simulation. These capabilities are defined as a collection of methods and daemons at the abstract class level. Examples of these are shown below.

```
;;;
;;;  After an instance of any model-component is created, it
is
;;;  asserted as a part of the current model instance.
;;;
(defmethod (model-component :after :new) ()
  (tell (pack* self '- (tell self :get instance-number))
        :assert part-of (tell model :get current-model)))

;;;
;;;   This method writes the current state of a model
component
```

```
;;;  to the specified output stream.  It is written as a
LISP
;;;  expression
;;;  so that when it is loaded back in, it will be evaluated
;;;  and set
;;;  the state of the object to the value specified.
;;;
(defmethod (model-component :write-state) (stream)
   (princ (pack* "$(tell " self " :set state "
                             (?attr self 'state) ")")
          stream)
   (terpri stream))
```

Coupled with the physical model is a representation of the events or actions that can be performed by the expert and the states that can be present for each of the equipment components. Event knowledge is an understanding of the potential actions that the expert may perform on the instantiated objects in the model, the effects of the action to the object acted upon, and the propagation of changes through the model along the relationships defined. In the VVL domain there are three actions that may be performed. Two affect the valve object, **open** and **close**. These actions change the state of the simulation and are used by the expert to identify the leak by isolating parts of the system. The third action is **inspecting** a gauge. This action does not affect the state of the underlying simulation but is used by the expert to guide the diagnosis process. This distinction between inspection actions and state-change actions on the part of the expert plays a key role in the induction of rules.

ACQUISITION PROCESS

The passive acquisition process is comprised of three phases,

1) **Observation,**
2) **Playback/Elicitation**, and
3) **Rule Generation.**

The next sections describe these phases in more detail.

Observation

In the observation phase, the system emulates a human knowledge-engineer. There are two sub-processes in this phase. The first is a domain analysis process. This process addresses the construction of domain knowledge for the qualitative simulation model. Contextual knowledge regarding the types and structures of the objects involved in the model, potential connectivity, relationships, and states are analyzed.

The domain analysis is performed as a precursor to observing the expert's actions during a simulation execution. In the example domain for this effort, the basic structural analysis of the model to be executed is constructed. General domain knowledge such as descriptions of valves, vacuum sources, etc. and their properties and behaviors have been "hand crafted" for this effort. Subsequent efforts could build on existing domain knowledge bases or previous efforts in constructing models and, as PaKAS is repeatedly applied, the corpus of reusable world knowledge will grow correspondingly.

The second sub-process is the observation of the expert's diagnosis action's during the simulation. This involves specifying the set of initial states and starting the simulation. The simulation technique used for the phase I efforts is a qualitative, discreet-event simulation technique. It is constructed via a collection of methods and daemons triggered by state changes in the model components.

For example, as the user opens or closes a valve, the effect of the vacuum source is propagated throughout the system based on the state of the valves. As the vacuum is propagated, the gauges' state are reset to nominal. In a second pass, the effect of the leak is propagated throughout the network staring at the leak source. As the leak is propagated, gauges are set to abnormal for those gauges affected by the leak.

Procedural knowledge is then gathered by observing the expert in action. The system accomplishes this by initializing a log file containing the initial state of all objects within the model. PaKAS then captures each action performed by the expert and saves them in the log file as well. Actions are stored are a triplet consisting of the actions performed, the object on which the expert performed the action, and the state of the object. In the case of inspection actions, the state information is the state of the object inspected returned by the system. For state-change actions, it is the new state of the object as set by the expert. As the system collects the actions performed by the expert, it builds a temporal network of action instances. These form a procedural network which provides an initial ordering of the tests and actions performed by the expert.

Playback/Elicitation

The elicitation phase involves playing back the actions performed by the expert and querying for goals, rationale, anticipated changes, etc. This directly parallels the behavior of a human knowledge engineer. In this phase, commonsense knowledge would be advantageous. As described by (Lenat 1986), it is our common corpus of knowledge that allows us to master new information and situations. The human knowledge engineer may rely on a basic understanding of vacuums in order to ascertain the interdependencies between the various components of a model and provide insight into what the expert may have been thinking when performing a particular action.

The PaKAS elicitation process is built around the playback of the simulation run performed by the expert. Playback is performed through the interpretation of the log file entries for the simulation execution which is being analyzed. The playback follows the process used by a human knowledge engineer when eliciting knowledge from an expert. After watching the expert perform their task, the knowledge engineer will "walk through" the expert's actions.

During the playback of the simulation, PaKAS asks the expert for information regarding the actions performed. These questions can be thought of as addressing several areas. These include goal information, "Why was an action performed," strategy assessment, "When an action is performed," rationale, "What was the purpose of the action," anticipations of the expert, "What was the expected outcome (if any) of the action," and others.

Rule Generation

The first step is the development of instantiation-specific rules. These rules are collections of inspection and action patterns which were performed for the specific model instantiation. These are represented as the literal pairs of inspections and actions. The system assumes that a series of state inspection actions followed by state-change actions constitutes a specific rule instance.

The rule developed is specific, however, not only to the model being investigated, but also specific to the state of the model at that point in time. Because of this, the model specific rules developed are limited in scope and not very useful as a general rule that could be applied to other model instances in the domain. They do, however, form a solid base which can be analyzed and provide a base to develop more general-purpose rules for the domain.

Induction is then applied to build general rules which would be applicable across multiple model instantiations within the same domain. The approach used for the induction process was to develop a set of generalized tests from the specific rules. These generalized tests were developed using an interactive, stepwise refinement. The initial approach utilized a four step process described below.

1) *Replace instance references with the class id and a variable name* - This step takes the specific instance inspections such as (CHECK GAUGE-1 ABNORMAL) and transforms it into (GAUGE ?X STATE ABNORMAL). The variable name X is associated with GAUGE-1 so that any further references to GAUGE-1 encountered will result in the use of X as the instance handle variable.

2) *Add equality test for the state observed* - This step inserts the boolean predicate "=" into the antecedent clause, yielding (GAUGE ?X STATE ABNORMAL). This example was adequately addressed using the "=" test. This will require expansion of the elicitation process in phase II to check for value ranges.

3) *Build relationship tests using both the inspected objects and the state-change objects* - This step adds antecedent clauses which relate the inspection objects and state-change objects. For example, (GAUGE ?X IS-CONNECTED-TO VALVE ?Y), is a generated relationship test for the fact that a gauge is connected to a valve. This step begins to refine the capabilities of the rule by defining configurations in which the rule is valid.

4) *Add equality test for the previous state of the object(s) acted upon by the user* - Finally, tests are added which test for the prior state of the objects acted upon by the user. For example, if the user opens VALVE ?Y, the test that would be added is (VALVE ?Y STATE CLOSED).

EXECUTION EXAMPLE

This section walks through an interaction with the PaKAS system. The PaKAS user interface is shown in figure 3. The simulation is started when the expert clicks on the "Simulation" menu choice at the top of the screen. At this point, a new simulation history file is opened for the model, a pipe section is chosen at random for failure,[4] the vacuum and leak effects are propagated through the network, all initial states are then logged to the simulation history file,

[4]Although devices as well as pipes can fail causing leaks, the phase I prototype was limited to pipes due to the funding and time constraints of a phase I SBIR.

PaKAS enters observation mode, and control is turned over to the expert.

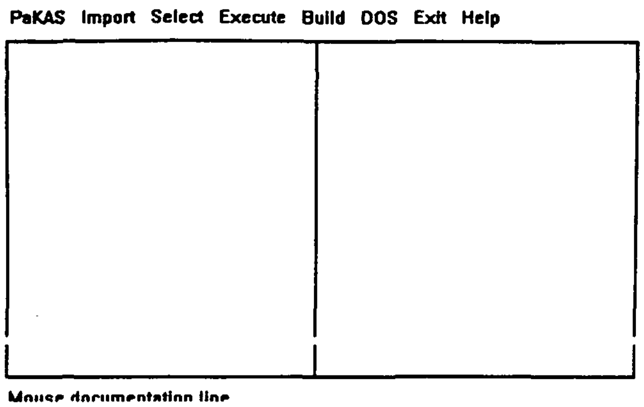

Figure 3. PaKAS User Interface

Once control is turned over to the expert, the expert then begins inspecting gauges and opening or closing valves until she believes the leak point has been identified. At this point, the simulation is completed by clicking again on the "Simulation" menu choice.

In our example, all valves are open at the start of the simulation. The expert clicks on gauges 1 and 3, finds both to be abnormal, then closes valve 2. Then gauge 1 is re-inspected and found to be nominal. At this point, the expert closes valve 3 and opens valve 2. Gauge 3 is re-inspected and found to be abnormal. The expert then click on "Simulation" to conclude this run having identified the leak to be in the pipe section connecting apparatus-2, valve-2, valve-3, and gauge-3.

At this point, PaKAS closes the simulation log file and returns to a command mode. An example of the simulation log file contents is shown in figure 4.

```
$(set-window 'history)
$(clear-screen)
$(center-line nil)
$(auto-crlf t)
Simulation run for: demo1
Date: 7/10/1990  --  16:25:26
Initializing model state...
$(tell GAUGE-1 :set state ABNORMAL)
$(tell GAUGE-2 :set state ABNORMAL)
$(tell GAUGE-2 :set state ABNORMAL)
$(tell EW-VALVE-1 :set state OPEN)
$(tell EW-VALVE-2 :set state OPEN)
$(tell NS-VALVE-1 :set state OPEN)
$(set-window 'model)
$(tell *current-model* :display)
$(set-window 'history)
Beginning elicitation playback...
$(CHECK GAUGE-1 ABNORMAL)
$(CHECK GAUGE-3 ABNORMAL)
$(CLOSE EW-VALVE-2)
$(CHECK GAUGE-1 NOMINAL)
$(CLOSE NS-VALVE-1)
```

```
$(OPEN EW-VALVE-2)
$(CHECK GAUGE-3 ABNORMAL)
Completion of elicitation playback...
```

Figure 4. Simulation Log File

In the elicitation phase, PaKAS opens the simulation log file and begins playing back the steps taken by the expert. The simulation log file is script of functions to be evaluated and text strings to be played back on the screen. The first few lines of the log file are initialization functions which are generated for all log files.

```
$(set-window 'history)
$(clear-screen)
$(center-line nil)
$(auto-crlf t)
```

These are followed by several text strings identifying the VVL/ICAT model instance for which the log file applies, the date and time of the simulation, and a descriptive message describing the initialization process.

```
Simulation run for: demo1
Date: 7/10/1990  --  16:25:26
Initializing model state...
```

The next few lines reset the state of all the objects within the model to their state at the start of the simulation.

```
$(tell GAUGE-1 :set state ABNORMAL)
$(tell GAUGE-2 :set state ABNORMAL)
$(tell GAUGE-2 :set state ABNORMAL)
$(tell EW-VALVE-1 :set state OPEN)
$(tell EW-VALVE-2 :set state OPEN)
$(tell NS-VALVE-1 :set state OPEN)
```

The system then redisplays the model on the screen and tells the expert that it is initiating the elicitation sequence.

```
$(set-window 'model)
$(tell *current-model* :display)
$(set-window 'history)
Beginning elicitation playback...
```

The remainder of the actions consist of the inspection or state change actions performed by the expert. The actions are logical broken into collections of inspection/action pairs. This forms a general equivalency with the production rule formalism.

```
$(CHECK GAUGE-1 ABNORMAL)
$(CHECK GAUGE-3 ABNORMAL)
$(CLOSE EW-VALVE-2)
$(CHECK GAUGE-1 NOMINAL)
```

```
$(CLOSE NS-VALVE-1)
$(OPEN EW-VALVE-2)
$(CHECK GAUGE-3 ABNORMAL)
Completion of elicitation playback...
```

The system restates the inspections and actions performed by the expert using template driven natural language. These are executed in sequence with the system eliciting responses from the expert at each set of inspection/action pairs. The expert is asked several basic questions regarding the intentions, goals, and anticipated outcome (if any) of the actions performed. An example of this interaction is shown in figure 5.

```
PaKAS> First you checked GAUGE-1 which was ABNORMAL.
PaKAS> Then you inspected GAUGE-3 which was ABNORMAL.
PaKAS> Next you CLOSED EW-VALVE-2.

PaKAS> Why did you CLOSE EW-VALVE-2?
Vince> The valve is roughly in the middle of the network.

PaKAS> What did you hope to find after closing EW-VALVE-2?
Vince> That GAUGE-1 had a NOMINAL reading.

PaKAS> Why would that be significant?
Vince> Because it would mean that the leak is after EW-VALVE-2.
```

Figure 5. Elicitation of Rationale

The generation phase then takes the collection of inspection/action pairs and develops model-specific rules. These rules are a literal conversion of the expert's actions into a model-specific production rule. Figure 6 shows an example of model-specific production rule instantiations..

```
RULE-1
     If
          (GAUGE-1 STATE EQ ABNORMAL)
          (GAUGE-3 STATE EQ ABNORMAL)
     Then
          (CLOSE EW-VALVE-2)

RULE-2
     If
          (GAUGE-1 STATE EQ NOMINAL)
     Then
          (CLOSE NS-VALVE-1)
          (OPEN EW-VALVE-2)

RULE-3
     If
          (GAUGE-3 STATE EQ ABNORMAL)
     Then
          (DONE)
```

Figure 6. Model-specific Production Rules

In viewing the model-specific rules it immediately becomes evident that, although a literal translation of expert actions into production rules for a specific model is certainly feasible, it is of little value. Firstly, the rules would be valid only for that particular model instantiation. This limitation alone would be enough to require further work. A second reason is the implicit reasoning and assumptions of the expert that are not evident by their actions alone.

An example can be found in figure 9 between RULE-1 and RULE-2. In the first rule, the expert checks two gauges and closes a single valve. In looking at the dependency relationship between the gauges and valve, we see that the valve is in between the two gauges on the dependency chain. When RULE-2 is initiated, the only inspection action is performed on GAUGE-1. Implicitly, what the expert has done is to bisect the dependency graph at EW-VALVE-2 by closing it. By then inspecting GAUGE-1, the expert immediately knows if the leak is before or after EW-VALVE-2.

Since GAUGE-1 had a NOMINAL reading, the leak is somewhere after EW-VALVE-2. So the expert has implicitly "marked" EW-VALVE-2 as the furthest known node in the network with no leak. This means that the leak is further along the dependency tree. Since there are two more pipe groups, the expert opens EW-VALVE-2 to allow the vacuum to propagate further down the dependency tree and closes NS-VALVE-1 to bisect the remaining pipe network.

At this point all that needs to be inspected is GAUGE-3. If it is NOMINAL then the leak is between NS-VALVE-1 and APPARATUS-3. Otherwise, it is in the pipes connecting EW-VALVE-2, NS-VALVE-1, GAUGE-3, and APPARATUS-2. The gauge does read nominal and the expert concludes the interaction.

Figure 7 below shows the CLIPS rules generated after applying the induction strategy to the instance rules.

```
(DEFRULE RULE-3
;;;  It was on the pipe network
       (GAUGE ?V1 ABNORMAL)
```

```
        (VALVE ?V2 OPEN)
        (?V2 CONNECTED-TO ?V1)
=>
        (ASSERT (VALVE ?V2 CLOSED))
;;; It bisected the network
)  ;;; END DEFRULE

(DEFRULE RULE-6
;;; It was between the closed valve and the vacuum source
        (GAUGE ?V4 NOMINAL)
        (VALVE ?V5 OPEN)
        (VALVE ?V2 CLOSED)
        (?V2 CONNECTED-FROM ?V4)
        (?V5 CONNECTED-FROM ?V4)
=>
        (ASSERT (VALVE ?V5 CLOSED))
        (ASSERT (VALVE ?V2 OPEN))
;;; to propagate the vacuum further
;;; to bisect the remaining section of pipe
)  ;;; END DEFRULE

(DEFRULE RULE-7
;;; to finalize the procedure
        (GAUGE ?V1 ABNORMAL)
=>
)  ;;; END DEFRULE
```

Figure 7. Generated CLIPS Rules

CONCLUSIONS

The use of a hybrid object-oriented knowledge representation system greatly simplified the representation and execution of both the domain model and the expert actions. Using a common representation allows the acquisition system to access domain information, object structures, expert events, and any related knowledge that may be entered into the system.

A common knowledge representation system for the individual components also allows the construction of corporate or commonsense knowledge that may be applied to subsequent domains. This capability eases the construction of new domains by being able to copy and modify structures from existing, related domains.

The representation of the domain within the knowledge representation system needs is being expanded in the phase II effort to encompass a wider range of world processes and events. The basic engine for representing the various inspection actions and state change actions should be extended to be similar to the event manager concept used in simulation languages and systems.

Finally, although CLIPS is the primary target, the promising results thus far makes this technology useful for other production languages and expert system shells. This aspect of the technology will be investigated as part of the commercialization effort of Phase III.

REFERENCES

Buchanan, Bruce G. *Some Approaches to Knowledge Acquisition.* Report No. KSL-85-38, Knowledge Systems Laboratory, Stanford University: Stanford, Calif., October, 1985.

Hayes-Roth, Frederick, and John McDermott. *Knowledge Acquisition from Structural Descriptions.* Report No. P-5910, Rand Corp.: Santa Monica, Calif., 1976.

Levesque, Hector J. "Foundations of a Functional Approach to Knowledge Representation." *Artificial Intelligence* **23**, Elsevier Science Publishers B.V.: New York, 1984. 155-212.

Lenat, Doug, Mayank Prakash, and Mary Shepherd. "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks." The AI Magazine, (volume, number, and date unknown). 65-85.

Wilkins, David C., William J. Clancey, and Bruce G. Buchanan. *Knowledge Base Refinement by Monitoring Abstract Control Knowledge.* Report No. KSL-87-01, Knowledge Systems Laboratory, Stanford University: Stanford, Calif., 1987.